Example:

Imitating Language Constructs

In this lecture, you will learn how to add custom methods that would look like Ruby language keywords. Examples of such methods: private, attr_reader, puts, raise, etc.

Background

```
# All these & other methods are defined in 'Kernel' module and included in Object class.
# All other classes subclass Object class. When called, they're eventually found in Kernel modul.

require 'file'                                                       # require('file')

raise NameError, "You've got an error!"                              # raise(NameError, "You've got an error!")

lambda { 'Also defined in Kernal module.' }                          # lambda(&block)

puts 'some text'                                                     # puts('some text')



# attr_reader, attr_accessor, etc. methods are defined in 'Module' class and thus,
# available in all modules and classes. Because, classes inherit from Module.

class MyClass
    attr_reader :foo                                                 # attr_reader(name)
end

class Person < ActiveRecord::Base
    validates :name, :presence => true                               # validates(:name, :presence => true)
end
```

How can we add our own language construct imitators?

- Define top level method
- Add method in Kernel module
  (or any ancestor of the class where you want to use it)

Option 1: Define top level method

```ruby
def string?(string)
    string.class == String
end


string? ""                                              # true
string? []                                              # false


class MyClass
    string? "foo"                                       # true

    def foo
        string? "foo"
    end
end

MyClass.new.foo                                         # true
```

Option 2: Add method to Kernel module

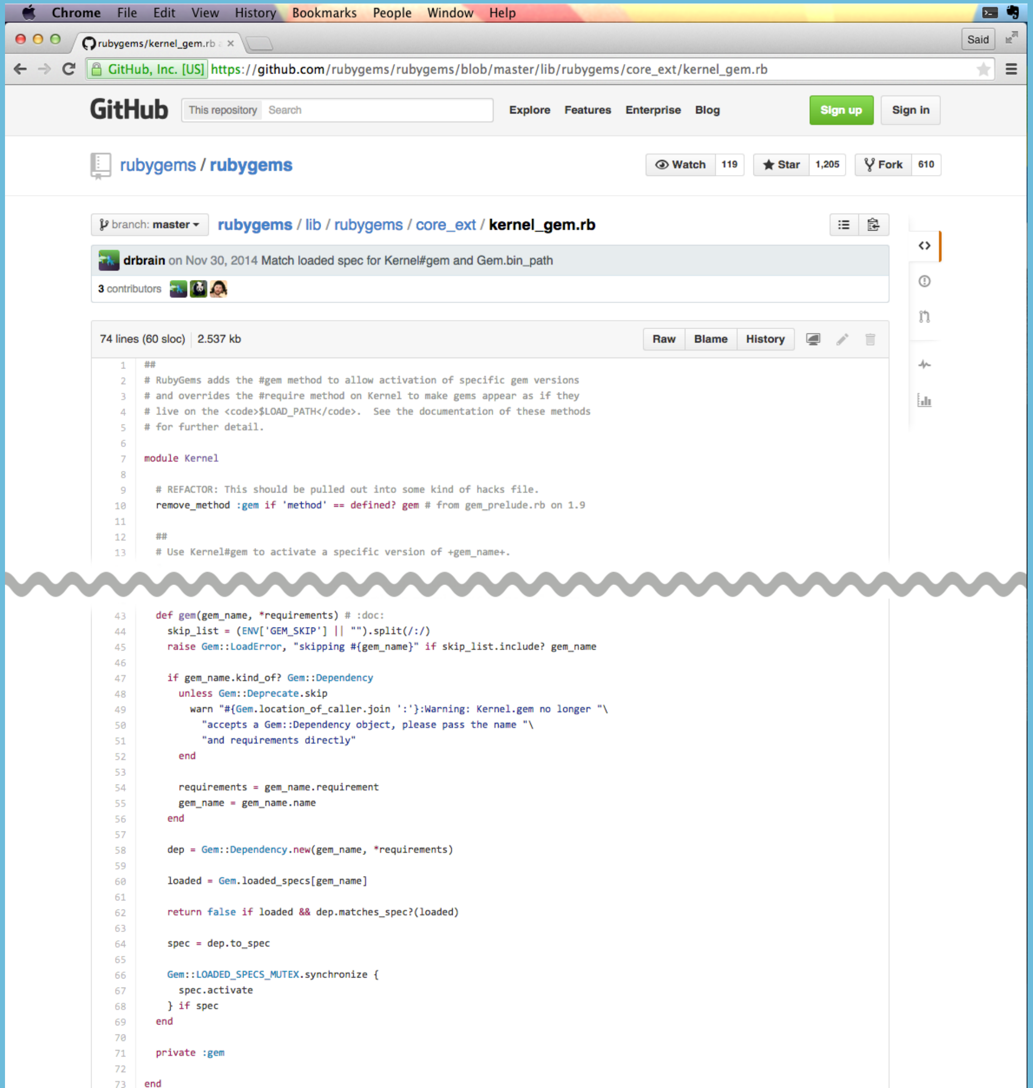
```
module Kernel
    def foo(arg)
        "Kernel#foo( #{arg} )"
    end
end


foo(42)                                                       # "Kernel#foo( 42 )"


class MyClass
    def bar
        foo 123
    end
end

obj.bar                                                       # "Kernel#foo( 123 )"
```

Ruby gem example


Chrome File Edit View History Bookmarks People Window Help
rubygems/kernel_gem.rb
GitHub, Inc. [US] https://github.com/rubygems/rubygems/blob/master/lib/rubygems/core_ext/kernel_gem.rb
GitHub
This repository Search
Explore Features Enterprise Blog
Sign up Sign in
rubygems / rubygems
Watch 119 Star 1,205 Fork 610
branch: master
rubygems / lib / rubygems / core_ext / kernel_gem.rb
Raw Blame History

drbrain on Nov 30, 2014 Match loaded spec for Kernel#gem and Gem.bin_path

3 contributors

74 lines (60 sloc) | 2.537 kb

```ruby
##
# RubyGems adds the #gem method to allow activation of specific gem versions
# and overrides the #require method on Kernel to make gems appear as if they
# live on the <code>$LOAD_PATH</code>.  See the documentation of these methods
# for further detail.

module Kernel

  # REFACTOR: This should be pulled out into some kind of hacks file.
  remove_method :gem if 'method' == defined? gem # from gem_prelude.rb on 1.9

  ##
  # Use Kernel#gem to activate a specific version of +gem_name+.
```

```ruby
  def gem(gem_name, *requirements) # :doc:
    skip_list = (ENV['GEM_SKIP'] || "").split(/:/)
    raise Gem::LoadError, "skipping #{gem_name}" if skip_list.include? gem_name

    if gem_name.kind_of? Gem::Dependency
      unless Gem::Deprecate.skip
        warn "#{Gem.location_of_caller.join ':'}:Warning: Kernel.gem no longer "\
          "accepts a Gem::Dependency object, please pass the name "\
          "and requirements directly"
      end

      requirements = gem_name.requirement
      gem_name = gem_name.name
    end

    dep = Gem::Dependency.new(gem_name, *requirements)

    loaded = Gem.loaded_specs[gem_name]

    return false if loaded && dep.matches_spec?(loaded)

    spec = dep.to_spec

    Gem::LOADED_SPECS_MUTEX.synchronize {
      spec.activate
    } if spec
  end

  private :gem

end
```

Rubygems defines gem method in 'Kernel' module.

```
gem "rails", ">= 4.2"
```

https://github.com/rubygems/rubygems/blob/master/lib/rubygems/core_ext/kernel_gem.rb

Example:

Assume you are writing a gem that allow users to send notifications to groups of users using their preferred method.

Your gem allows users to send notifications using this syntax:

```
notify :admins, "Long process ended!"
```

Users of your gem, should be able to run this code from anywhere in their code.

Adding custom “notify” feature

```ruby
notify :admins, "Long process ended!"


module Kernel
    def notify(who, message = "")
        # Get list of all 'admins'. Send
        # 'message' using each user's prefered method.
    end
end
```